Understanding Spark Performance in Hybrid and Multi-Site Clouds

Roxana Ioana Roman
University of Rennes 1
Rennes, France
roxana-ioana.roman@irisa.fr

Bogdan Nicolae
IBM Research
Damastown, Mulhuddart
Dublin, Ireland
bogdan.nicolae@acm.org

Alexandru Costan
IRISA / INSA de Rennes
Campus de Beaulieu
Rennes, France
alexandru.costan@irisa.fr

Gabriel Antoniu
Inria Rennes
Campus de Beaulieu
Rennes, France
gabriel.antoniu@inria.fr

ABSTRACT

Recently, hybrid multi-site big data analytics (that combines on-premise with off-premise resources) has gained increasing popularity as a tool to process large amounts of data on-demand, without additional capital investment to increase the size of a single datacenter. However, making the most out of hybrid setups for big data analytics is challenging because on-premise resources can communicate with off-premise resources at significantly lower throughput and higher latency. Understanding the impact of this aspect is not trivial, especially in the context of modern big data analytics frameworks that introduce complex communication patterns and are optimized to overlap communication with computation in order to hide data transfer latencies. This paper contributes with a work-in-progress study that aims to identify and explain this impact in relationship to the known behavior on a single cloud. To this end, it analyses a representative big data workload on a hybrid Spark setup. Unlike previous experience that emphasized low end-impact of network communications in Spark, we found significant overhead in the shuffle phase when the bandwidth between the on-premise and off-premise resources is sufficiently small.

Keywords

Spark, data management, multi-site processing

1. INTRODUCTION

Due to exploding data sizes and the need to combine multiple data sources, single-site processing becomes insufficient as big data analytics applications can no longer be accommodated within a single datacenter. To enable such complex processing, a promising approach consists in simultaneously provisioning resources on *multiple datacenters* at different geographical locations from the same cloud provider (multi-site) or combine permanent resources on a private datacenter (or long-term leased cloud resources) with temporary, off-premise resources (hybrid setups). This may have several benefits: resilience to failures, better locality (e.g., by moving computation close to data or viceversa), elastic scaling to support usage bursts, user proximity through content delivery networks, etc. In this context, sharing, disseminating and analyzing the data sets may result in frequent large-scale data movements across widely distributed sites. Studies show that the inter-datacenter traffic is expected to triple in the following years [8, 11]. At the same time, this unprecedented geographical distribution of data and resources raises new challenges. Besides the complexity introduced by the heterogeneity of resources from hybrid deployments, the network constitutes a major bottleneck. Cloud datacenters are interlinked with *high-latency, low-throughput wide-area networks*, making inter-site data transfers up to an order of magnitude slower than intra-site data transfers.

At the lower layers of the data analytics stack, significant strides have been made in providing vastly scalable data management solutions that also achieve high availability via remote WAN resource provisioning. NoSQL systems like Cassandra [10] or Riak [2] are able to gracefully scale up at geographically distant locations, albeit at reduced replica consistency. Even some databases, like Google's F1 [17] have managed to scalably process queries and transactions across sites. Data transfers tools like JetStream [20] exploit the inherent network parallelism of the hybrid setups and enable fast data movements.

However, moving up the data analytics stack, large-scale analytics frameworks like Hadoop [21] or Spark [24] have not achieved the same level of scalable, cross-datacenter implementation. Despite key design choices to leverage data locality (such as coupling the analytics runtime with the storage layer to be able to schedule the computation close to the data), many inherently difficult communication patterns (in particular concerning tightly coupled all-to-all communication that happens during collective data aggregation through reduce operations) have led to a situation where users avoided the distribution of the same application on a hybrid setup, preferring to use hybrid setups in order to accelerate a set of independent or loosely coupled applications.

This paper aims to study how the weak link of a hybrid setup impacts big data analytics applications that span beyond a single cluster, as a first step in understanding how to optimize the runtime middleware for such configurations. In particular, we focus on Spark as a representative big data analytics framework, due to the recent interest in its capabilities (both functional and non-functional) over Hadoop MapReduce and its ecosystem. Our goal is to identify and

explain the performance bottlenecks under a hybrid scenario for Spark when compared with a single cluster scenario, where previous performance studies already showed that the CPU is increasingly becoming a bottleneck versus the disk or network [14].

We summarize our contributions as follows:

- We identify and discuss several challenges of running a big data analytics application over a Spark deployment that spans multiple sites in a hybrid setup (Section 3.2).
- We design an experimental setup that emulates an easily configurable real-life hybrid setup in order to facilitate the evaluation of multiple configurations under controlled settings (Section 4.1).
- We evaluate Spark in a series of extensive experiments where we found significant job completion time overhead due to low inter-cluster bandwidth, which is mostly due to slower transfers during data shuffling (Sections 4.2 and 4.3).

2. RELATED WORK

While extensive research efforts have been dedicated to optimize the execution of big data analytics frameworks, there has been relatively less progress on identifying, analyzing and understanding the performance issues of these systems in a hybrid setup.

Execution optimization. Since the targeted applications are mostly data-intensive, a first approach to improving their performance in hybrid environments is to make *network optimizations*. Multi-hop path splitting solutions [7] replace a direct TCP connection between the source and destination by a multi-hop chain through some intermediate nodes. Multi-pathing [16] employs multiple independent routes to simultaneously transfer disjoint chunks of a file to its destination. JetStream [20] is specifically targeting geographically distributed data transfers: building on the network parallelism, the throughput can be enhanced by routing data via intermediate nodes chosen to increase aggregate bandwidth. These solutions come at some costs: under heavy load, per-packet latency may increase due to timeouts while more memory is needed for the receive buffers. On the other hand, end-system parallelism can be exploited to improve utilization of a single path. This can be achieved by means of parallel streams [6] or concurrent transfers [13]. Although using parallelism may improve throughput in certain cases, one should also consider system configuration since specific local constraints (e.g., low disk I/O speeds or over-tasked CPUs) may introduce bottlenecks. More recently, a *hybrid* approach was proposed [22] to alleviate from these. It provides the best parameter combination (i.e. parallel stream, disk, and CPU numbers) to achieve the highest end-to-end throughput between two end-systems. *Storage optimizations* try either to better exploit disk locality [19] or simply to eliminate the costly disk accesses by complex in-memory caches [23, 12]. In both cases, the resulting aggregated uniform storage spaces will lag behind in widely distributed environments due to the huge access latencies.

Performance analysis and prediction. The vast majority of research on this field focuses on the Hadoop framework, since, for more than a decade, this has become the de-facto industry standard. The problem of how to predict completion time and optimal resource configuration for a MapReduce job running on a hybrid cloud was proposed in [**?**]. To this end, the work introduces a methodology that combines analytical modelling with micro-benchmarking to estimate the time-to-solution in a given hybrid configuration. The problem of disproportionately long-running tasks, also called stragglers, has received considerable attention, with many mitigation techniques being designed around *speculative execution* [4, 5]. This approach waits to observe the progress of the tasks of a job and launches duplicates of slower tasks. However, speculative execution techniques have a fundamental limitation in hybrid setups: waiting to collect meaningful task performance information and then remotely scheduling tasks limits drastically their reactivity. Other studies focus on the *partitioning skew* [9] which causes huge data transfers during the shuffle phases, leading to significant unfairness between nodes. More recent performance studies specifically target Spark [14]. However, the authors are only interested in the impact of several parameters (CPU, network, disk) on applications running within a single datacenter.

Overall, most of the previous work typically focuses on some specific issues of big data frameworks: either a single cluster, or, if the work targets a hybrid setup, the focus falls on specific low-level issues that are not necessarily well correlated with the higher level design. It is precisely this gap that we aim to address in this work by linking bottlenecks observed through low level resource utilization with high-level behavior in order to better understand performance in a hybrid setup.

3. BACKGROUND

3.1 The Spark Architecture

Spark is a big data analytics framework that facilitates the development of multi-step data pipelines using directly acyclic graph (DAG) patterns. The user implements a driver program that describes the high-level control flow of the application, which relies on two main parallel programming abstractions: (1) *resilient distributed datasets* (RDDs), which describe the data itself; and (2) parallel operations on the RDDs.

An RDD is a read-only, resilient collection of objects partitioned across multiple nodes that holds provenance information (referred to as *lineage*) and can be rebuilt in case of failures by partial recomputation from ancestor RDDs. An RDD can be created in three ways: (1) by using the implicit partitioning of an input file stored in the underlying distributed file system (e.g., HDFS); (2) by explicit partitioning of a native collection (e.g. array); or (3) by applying a transformation to an already existing RDD. Furthermore, each RDD is by default *lazy* and *ephemeral*. The lazy property has the same meaning as in functional programming and refers to the fact that an RDD is computed only when needed. Ephemeral refers to the fact that once an RDD actually got materialized, it will be discarded from memory once it was used. However, since RDDs might be repeatedly needed during computations, the user can explicitly mark them as persistent, which moves them in a dedicated cache for persistent objects.

There are three major transformations possible on RDDs: *reduce*, *collect* and *foreach*. Reduce has the same meaning

as the reduce phase of MapReduce: it represents a same key aggregation of values according to a custom function. Collect gathers the elements of an RDD into a native collection on the driver program (e.g. array). Foreach is similar to the map phase of MapReduce: it applies a parallel transformation on all elements of the RDD.

These transformations seem to emulate the expressivity of the MapReduce paradigm overall, however, there are two important differences: (1) due to the lazy nature, maps will only be processed before a reduce, accumulating all computational steps in a single phase; (2) RDDs can be cached for later use, which greatly reduces the need to interact with the underlying distributed file system in more complex workflows that involve multiple reduce operations.

3.2 Challenges of running Spark in a Hybrid Setup

Since the early days of big data analytics, MapReduce relied heavily on a key idea to conserve network bandwidth in order to achieve large-scale scalability on commodity clusters: taking advantage of *data locality*. More specifically, the storage layer is co-located with the runtime on the same nodes and is designed to expose the location of the data blocks, effectively enabling the scheduler to bring the computation close to the data and avoid a majority of the storage-related network traffic. This basic idea is also adopted by modern big data analytics frameworks, including Spark.

Nevertheless, despite significant progress in exploiting data locality, runtimes still face a major challenge: efficient aggregation of intermediate results during the reduce phase. While transparent for the user, from whose viewpoint the reduce is a collective operation agnostic of the data distribution, eventually all the burden of synchronization and collection of relevant data pieces (which is why Hadoop MapReduce and Spark are referred to as *data shuffling*) falls on the runtime. At scale, this involves concurrent remote transfers of significant data amounts between the nodes, which unsurprisingly stresses the networking infrastructure.

However, in a hybrid setup, the link between the on-premise infrastructure and the external off-premise resources is typically of limited throughput and high latency. Thus, communication between on-premise and off-premise resources may create a network bottleneck both with respect to individual data transfers (i.e. low end-to-end throughput) as well as during parallel data transfers (i.e. low aggregated throughput despite sufficient end-to-end throughput). For the rest of this paper, we refer to the link between the on-premise and off-premise resources and these I/O bottlenecks it may cause simply as the "weak link".

This weak link affects two aspects. First with respect to input data, if the hybrid computation was specifically initiated to combine on-premise custom input data with off-premise public data or if the computation is part of a multi-stage job where the input data is already cached in memory, then the off-premise resources can benefit from data locality out-of-the-box. Otherwise, if input data needs to be shipped to the off-premise resources, this can happen either before the computation starts or during the computation using asynchronous techniques (e.g. HDFS rebalancing [**?**]). How to customize such asynchronous techniques to optimize the use of network bandwidth at the same time as Spark is using the network bandwidth itself is a direction we plan to explore in future work.

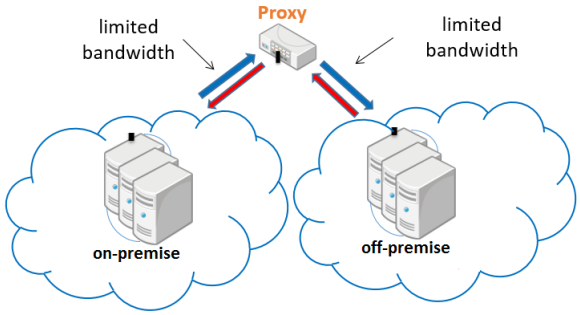


Figure 1: Architecture of the experimental setup

For the purpose of this work, we assume the input data is already distributed between the on-premise and off-premise resources in order to isolate the impact of the weak link on the data shuffling phase.

4. EXPERIMENTAL EVALUATION

This section presents a preliminary experimental evaluation using *WordCount*, a popular benchmarking workload initially used for MapReduce and ported to Spark, which is used to count the number of occurrences of each unique word in a large, distributed input dataset.

4.1 Setup

For our experiments we use Grid5000 [**?**], a large-scale experimental testbed comprising nine sites in France. We reserved nodes on the Rennes cluster, which comprises nodes interconnected with 10 Gigabit Ethernet. Each node has two Intel Xeon E5-2630v3 (2.4GHz) processors (2x6 cores), 128 GB RAM, 500 GB HDD. On these nodes we deployed Apache Spark 1.3.1 in standard configuration: one node is the *master*, while the rest of the nodes are configured as *workers*. Each worker is configured to use 45 GB memory and the maximal number of cores. We used the standalone cluster manager. Furthermore, we use HDFS as the underlying storage layer: all worker nodes are also configured to run an HDFS DataNode, while the master is configured to run an HDFS NameNode.

We split the worker nodes into two groups, which simulates a mix of on-premise and off-premise resources. This separation is transparent for Spark and HDFS, both of which treat the whole deployment as if it were a single cluster. In order to emulate the weak link between the on-premise and off-premise worker nodes, we configured an additional node to act as a proxy that forwards all traffic between the two groups, as shown in Figure 1. Then, using the *netem* kernel module we place configurable limits for this traffic, which enables us to vary both aspects (mentioned in Section 3.2) of the weak link for the study.

Note that this setup is very close to real life production cloud systems based on *OpenStack* [**?**]: in a typical configuration based on *neutron* (the standard OpenStack network management service), there is a *network node* responsible for network management. The VM instances are configured to directly communicate with each other via the links of their compute node hosts. However, all communication with the outside of the cloud is routed through the network

node. Thus, in a real-life hybrid cloud setup that involves two OpenStack deployments, any communication between on-premise and off-premise VMs will pass through the network nodes, which become the weak link.

During the experiments, we monitor all nodes in order to collect fine-grain system-level information regarding CPU, memory, disk and network interface. For this purpose, we run a *sar*-based daemon that logs all information at 5 second intervals. This information is then processed and aggregated after the experiments to produce the graphs presented in Section 4.

4.2 Methodology

The goal of the experiments is to study the end-impact of the weak link in a hybrid setting under variable scalability requirements. To this end, we vary two parameters: (1) the total number of nodes; and (2) the capacity of the weak link. We generate various combinations of (1) and (2) to create hybrid setups that we then compare with a *baseline* setup where all nodes are considered on-premise and form a single group that does not use a proxy or suffer the effects of a weak link. We decided for a single strategy to split the nodes into the on-premise and off-premise groups: half of the nodes are in one group and the rest in the other group. This decision was motivated by the need to minimize the bisection bandwidth of the nodes in such way that it coincides with the configurable capacity of the weak link. Using this strategy, the all-to-all communication pattern during the shuffle phase will potentially suffer the most negative impact due to the weak link, which emphasizes a worst case scenario. Thus, for the rest of this paper, we refer to a hybrid setup only by the total number of nodes, implicitly assuming that the on-premise and off-premise groups are of equal size.

We use a total of nine hybrid configurations, obtained as a variation of the number of nodes (6, 10, 20) and the capacity of the weak link (1 Gbps, 50 Mbps, 30 Mbps). In addition, three baseline configurations without a weak link corresponding 6, 10, 20 nodes are used. The experiments target weak scalability, with a fixed size of the input data of 25 GB per node. To prepare the input for WordCount, we downloaded the Wikipedia archive and uploaded it into a fresh HDFS deployment before running the corresponding WordCount Spark job. In this regard, the wiki input was replicated to reach the corresponding data size needed for a configuration. The default task parallelism is set to 1000, which is within the recommended values (i.e. at least double the total number of available cores to avoid starvation of short tasks).

4.3 Results

The completion time of the WordCount job for each of the configurations is depicted in Figure 2. As it can be observed, the WordCount application scales well with increasing number of workers in the baseline case, which is an expected behavior in the single-cluster scenario. What is interesting to observe is the relatively low impact of limiting the bandwidth to 1 Gbps: there is hardly any visible slowdown despite an order of magnitude less capacity compared to the baseline (i.e., 10 Gbps). However, further decreases in the capacity of the weak link begin to have a visible impact: for 6 workers a limitation of 50 Gbps increases the runtime by 50%, while a limitation of 30 Gbps increases the runtime more than double. This trend is observable for 20 workers too: this time, due to the fact that a larger amount of nodes share the weak link, the runtime increases by more than double and, respectively, four-fold.

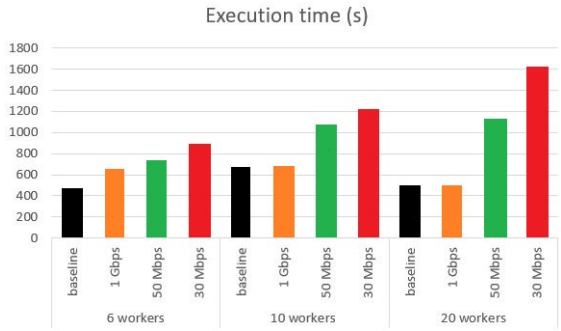
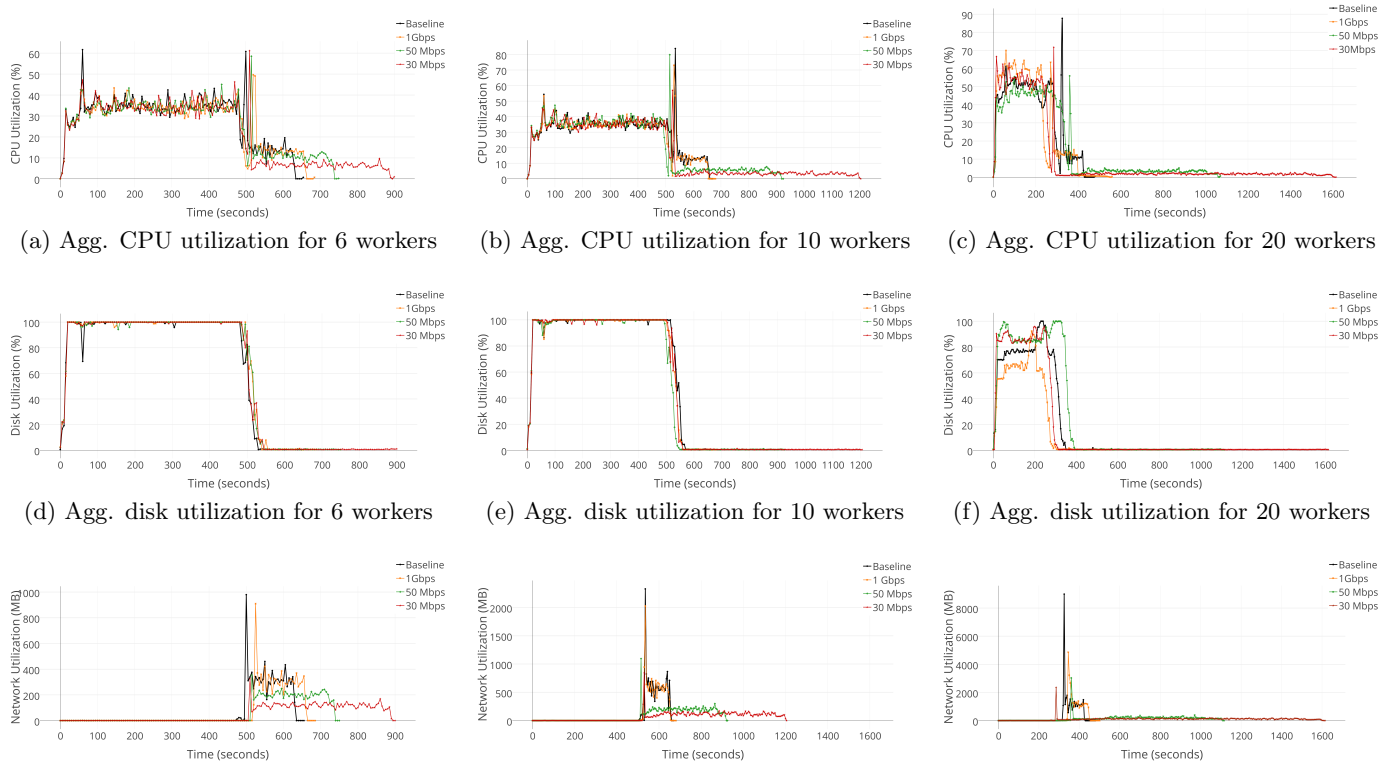


Figure 2: Completion time for Spark WordCount under variable cluster size and weak link capacity

To explain this behavior, we depict in Figure 3 the system level CPU, disk and network utilization during the runtime of WordCount for each configuration, grouped by number of workers. Note that the results are aggregated from all nodes. For CPU and disk, this aggregated value is given in percentage as the average utilization of all workers. For the network, the aggregated value corresponds to the sum of the traffic reported by all nodes and is given in MB/s.

As can be observed, there is a high disk utilization in the first phase of the computation, which corresponds to reading the input data from HDFS and constructing an in-memory RDD out of it. Once this is done however, disk utilization remains negligible. This trend remains stable regardless of number of nodes or capacity of the weak link. With respect to the CPU utilization during this first phase, it can be observed that it remains at around 50%, which is surprisingly high given the fact that this phase is dominated by disk activity. This however is explained by the fact that Spark initiates a lazy computation that interleaves the disk activity to build the input RDD with the subsequent "foreach" transformations. Since the reduce operation needs all foreach operations to finish before starting, there is no network activity visible during this phase, which also hints at the fact that data locality is well leveraged, i.e. remote HDFS transfers are avoided.

More interesting behavior is observable in the second phase, when the data shuffling is performed. This time, disk activity is negligible and CPU activity correlates well with network activity, which means that an increasing limit on the weak link introduces delays in the transfers of intermediate shuffle blocks, whose impact cause the CPU to wait for data. While this effect is not visible when comparing the baseline with the 1 Gpbs case, the effect becomes clear for 50 Mbps and 30 Mbps: the CPU and the network utilization stay at a lower level for longer, which explains the increase in overall runtime. Also interesting to note is that there is an initial burst of all-to-all network communications right at the beginning of the data shuffling, after which the network traffic stabilizes as the aggregation computation overlaps with the subsequent network transfers. Thus, a limit on the weak link not only affects the ability to overlap the CPU with the network I/O, but it also delays the moment when the aggregation can start.



(a) Agg. CPU utilization for 6 workers (b) Agg. CPU utilization for 10 workers (c) Agg. CPU utilization for 20 workers

(d) Agg. disk utilization for 6 workers (e) Agg. disk utilization for 10 workers (f) Agg. disk utilization for 20 workers

(g) Agg. network utilization for 6 workers (h) Agg. network utilization for 10 workers (i) Agg. network utilization for 20 workers

Figure 3: Aggregated system level resource utilization for Spark WordCount under variable cluster size and weak link capacity

5. CONCLUSIONS AND FUTURE WORK

In this work, we presented a preliminary study of the impact of using heterogeneous network links in hybrid setups that involve the use of resources from multiple clusters (i.e. augment on-premise resources with external, off-premise resources) when running large scale big data analytics based on Spark.

Unlike previous work that emphasized the relatively low impact of network and disk performance on the completion time for Spark workloads on a single cluster (meaning the CPU tends to be the bottleneck), we have discovered that the network can become a bottleneck in a hybrid setup, if the weak link between the on-premise and off-premise resources is sufficiently small (i.e. in the order of tens of Mbps). This impact can become dramatic when an increasing number of nodes need to communicate over a slow weak link, with observed completion time several times higher. While many geographically separated sites can communicate well beyond 1 Gbps and there is a tendency to increase the bandwidth of the weak link, we believe the observed bottlenecks will remain relevant for two reasons: (1) the number of nodes involved in a hybrid Spark deployment will be much larger, thus more nodes will share the weak link; (2) multiple other (Spark or completely different) applications and users will share the weak link concurrently, which makes it hard to dedicate a predefined bandwidth for a particular application.

As future work, we plan to explore the interleaving between CPU utilization and network transfers during the Spark reduce phase in greater detail, in order to better understand the behavior observed in our experiments, both with respect to the initial spike and the subsequent stabilized pattern. In particular, we plan to instrument Spark to understand what network traffic is generated by what worker. Based on this information it may be possible to better understand when delays are caused by the weak link and whether there is any skew in the transfers of the intermediate shuffle data. This in turn can help designing better scheduling algorithms specifically for a hybrid setup (e.g., assign the tasks to the workers such that most traffic happens within the same group and the traffic between on-premise and off-premise workers is minimized). Also, another interesting direction is related to the irregularity of the weak link: based on the assumption that it is shared with other applications and users, we plan to simulate a fluctuating capacity of the weak link in order to understand its impact.

6. REFERENCES

[1] Azure Succesful Stories. http://www.windowsazure.com/en-us/case-studies/archive/.

[2] Riak KV. http://basho.com/products/riak-kv/.

[3] Windows Azure. http://windows.azure.com.

[4] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica. Effective straggler mitigation: Attack of the clones. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*,

nsdi'13, pages 185–198, Berkeley, CA, USA, 2013. USENIX Association.

[5] G. Ananthanarayanan, M. C.-C. Hung, X. Ren, I. Stoica, A. Wierman, and M. Yu. Grass: Trimming stragglers in approximation analytics. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 289–302, Berkeley, CA, USA, 2014. USENIX Association.

[6] T. J. Hacker, B. D. Noble, and B. D. Athey. Adaptive data block scheduling for parallel tcp streams. In *Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*, HPDC '05, pages 265–275, Washington, DC, USA, 2005. IEEE Computer Society.

[7] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, I. Foster, and J. Saltz. Using overlays for efficient data transfer over shared wide-area networks. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 47:1–47:12, Piscataway, NJ, USA, 2008. IEEE Press.

[8] T. Kosar, E. Arslan, B. Ross, and B. Zhang. Storkcloud: Data transfer scheduling and optimization as a service. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, Science Cloud '13, pages 29–36, New York, NY, USA, 2013. ACM.

[9] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: Mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 25–36, New York, NY, USA, 2012. ACM.

[10] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[11] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. *SIGCOMM Comput. Commun. Rev.*, 41(4):74–85, Aug. 2011.

[12] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, memory speed storage for cluster computing frameworks. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 6:1–6:15, New York, NY, USA, 2014. ACM.

[13] W. Liu, B. Tieman, R. Kettimuthu, and I. Foster. A data transfer framework for large-scale science experiments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 717–724, New York, NY, USA, 2010. ACM.

[14] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, pages 293–307, Berkeley, CA, USA, 2015. USENIX Association.

[15] L. Pineda-Morales, A. Costan, and G. Antoniu. Towards multi-site metadata management for geographically distributed cloud workflows. In *Proceedings of the IEEE Cluster Conference*, Cluster'15, pages 1–10, 2015.

[16] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley. Data center networking with multipath tcp. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 10:1–10:6, New York, NY, USA, 2010. ACM.

[17] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, J. Cieslewicz, I. Rae, T. Stancescu, and H. Apte. F1: A distributed sql database that scales. In *VLDB*, 2013.

[18] A. Thomson and D. J. Abadi. Calvinfs: Consistent wan replication and scalable metadata management for distributed file systems. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 1–14, Berkeley, CA, USA, 2015. USENIX Association.

[19] R. Tudoran, A. Costan, G. Antoniu, and H. Soncu. Tomusblobs: Towards communication-efficient storage for mapreduce applications in azure. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 427–434, Washington, DC, USA, 2012. IEEE Computer Society.

[20] R. Tudoran, O. Nano, I. Santos, A. Costan, H. Soncu, L. Bougé, and G. Antoniu. Jetstream: Enabling high performance event streaming across cloud data-centers. In *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, DEBS '14, pages 23–34, New York, NY, USA, 2014. ACM.

[21] T. White. *Hadoop: The Definitive Guide*. Yahoo! Press, USA, 2010.

[22] E. Yildirim and T. Kosar. Network-aware end-to-end data throughput optimization. In *Proceedings of the first international workshop on Network-aware data management*, NDM '11, pages 21–30, New York, NY, USA, 2011. ACM.

[23] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *HotCloud'10*, pages 10–10, Boston, USA, 2010. USENIX.